AFRL-IF-RS-TR-2002-87
Final Technical Report
April 2002

THE ARIEL DISTRIBUTED PROGRAMMING PROJECT: SECURE EXECUTION OF MOBILE PROGRAMS

University of California at Davis

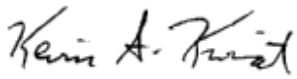
Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. F208

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-87 has been reviewed and is approved for publication.

APPROVED: 

KEVIN A. KWIAT
Project Engineer

FOR THE DIRECTOR: 

WARREN H. DEBANY, Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	APRIL 2002	Final May 97 – Nov 00

4. TITLE AND SUBTITLE
THE ARIEL DISTRIBUTED PROGRAMMING PROJECT: SECURE EXECUTION OF MOBILE PROGRAMS

5. FUNDING NUMBERS
C - F30602-97-1-0221
PE - 62301E
PR - F208
TA - 71
WU - 03

6. AUTHOR(S)
Raju Pandey

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of California at Davis
Computer Science Department
Davis California 95616

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFGA
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2002-87

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Kevin A. Kwiat/IFGA/(315) 330-1692/Kevin.Kwiat@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

Initially, the primary goal of the project was to develop techniques for making systems safe from mobile programs. During the course of the project, several important results were achieved in security, resource scheduling, runtime system design, and distributed programming environments. These results include an access control specification language, novel tools and techniques for enforcing security policies, safe threads package, resource scheduling algorithms for protecting against denial of service attacks, dynamic Java virtual machine, dynamically configurable security policy systems, and an adaptive distributed programming environment.

14. SUBJECT TERMS
Mobile Code, Mobile Programs, JAVA, JVM, Security, Distributed Systems

15. NUMBER OF PAGES
13

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

Abstract

This document presents the final report of the University of California, Davis-based project, officially titled "Secure execution of mobile programs," primarily known as the Ariel project. The primary goal of the initial project was to develop techniques for making systems safe from mobile programs. During the course of the project, several important results in security, resource scheduling, runtime system design, and distributed programming environments were achieved. These results include an access control specification language, novel tools and techniques for enforcing security policies, safe threads package, and resource scheduling algorithms for protecting against denial of service attacks, dynamic Java virtual machine, dynamically configurable security policy systems, and an adaptive distributed programming environment.

1. Overview

The overall goal of the Ariel project at UC Davis is to develop an adaptive, extensible and secure distributed programming environment. The environment supports adaptivity and extensibility by allowing components of programs to migrate freely from one name space to another namespace. While appealing both from system design and extensibility points of view, programming environments that support free movement of mobile programs are extremely vulnerable to aberrant execution behaviors of mobile programs. Since mobile program components run within the same name space as the runtime system, many of the traditional protection mechanisms such as address space containment and selective code loading no longer apply. Mobile programs, therefore, have the ability to maliciously disrupt the execution behavior of runtime systems by interfering with their execution, by interfering with the execution of other programs within the name space, by using unauthorized resources, by over-using resources, and by denying resources to other programs. The primary focus on the DARPA grant was to develop and implement system techniques [5] that will ensure safe and secure execution of external programs. The proposal had primarily focused on the following key objectives:

- Development of an access constraint specification language,
- Development of compilers and tools that enforce security policies through runtime program modification, and
- Development of optimization and verification techniques for ensuring secure execution of mobile programs.

Over the course of the proposal, our work evolved slightly away from our original objective. We ended up developing several new research components that were not part of the original proposal, but that, we believe, form essential elements of a secure distributed runtime system. This also means that we could not address some of the issues that we had promised to look at in the original proposals. This primarily involved developing optimization and verification techniques for ensuring secure execution of mobile programs. Below, we summarize the original and new approaches that we developed during the course of the project.

- Declarative language for specifying security policies;
- Binary editing tool for modifying Java byte code programs;
- Security enforcement tool that enforces security policies by transforming Java mobile programs;
- Safe thread package;
- Resource scheduling mechanisms for protecting mobile runtime environment

from denial of service attacks;

- Dynamically evolving Java Virtual Machine; and

- Support for dynamic security policies.

Administrative note: We had a slower rate of expenditures during the first year of the project because of problems in hiring graduate students and research staff. This resulted in a reduction of funding during the third year of the project. We were also granted a six month extension at the end of the third year. The extension helped us tie many loose ends and achieve new results. While we did not complete all of the tasks that we had set out to do, we achieved the central goal of the project along with many new interesting results.

Outline: The rest of this report is organized as follows: Our research in providing a secure environment for mobile programs focused on developing techniques for three security aspects: access control, safety, and safe resource allocation. We describe enforcement of access control in Section 2.1, safety in multi-threaded system in Section 2.2, and secure resource scheduling in Section 2.3.

Over the course of the research, we realized that much of our techniques were static in nature, and could not be changed once the system is in operation. This is a major flaw in most existing systems as they cannot respond to attacks dynamically by re-orienting their security policies, or even taking away specific privileges. This observation led to the development of system and security techniques that are truly dynamic in nature, and can evolve to meet emerging security needs. In Section 2.4, we first describe a dynamic Java virtual machine that we developed. We then describe a dynamic security infrastructure that can respond to changes in security policies dynamically.

We describe our technology transfer strategy in Section 4. We conclude by listing the publications.

2. Results

2.1 Enforcement of Access Control Policy

In this section, we give a brief summary of our approach to enforcing access control. The details can be found in [11, 12].

We have developed an *access control policy language* and a set of tools that enforce security policies on a mobile program. We describe them briefly:

- The *access constraint specification language* is used to specify constraints over accesses to resources. Our policy language is a simple declarative language of the form:

deny (C1.M1 –> C2.M2) when B

The semantics of this expression is that method *M*1 of class *C*1 cannot invoke method *M*2 of class *C*2 when *B* is true. The Boolean statement can be based on the parameters of the methods or the state of the system. The language also describes mechanisms for specifying constraints over a group of classes, an inheritance model, and mechanisms for overriding constraints.

- Security policies are enforced through tools by integrating the constraint checking code into the mobile program and resource codes. As a program, *P*, arrives at host, *H*1, Access Constraint Enforcement Tools examine *P*, local access constraints, and resource definitions to determine various access relationships. The tools then generate a set of constraint checking code which it patches into *P* and the resource definitions. The modified program then executes and accesses resources through the constraint checking code. Executions of this code ensures that *P* can access a resource only if site-specified constraints are true.

We have implemented the specification language and tools for Java-based mobile programs. We have also built a framework for defining and loading popular security models as security policy libraries.

2.2 Safety

We are building a multi-threaded distributed programming environment in which every mobile program is mapped to a single thread of execution. Our focus here was on developing an execution environment that protects the runtime system and the mobile programs from each other. Further, since data sharing among mobile programs may be dynamic and flexible, the execution environment must support sharing mechanisms that can be customized dynamically to reflect these sharing patterns. Thus, we need a protection mechanism that is flexible and provides dynamic sharing among threads of execution at the user level.

We have built a threads package, called *Safe Threads* that supports the notion of threads whose stacks and data elements are completely protected. The thread package contains a novel mechanism for specifying flexible and dynamic sharing and protection among threads. In this approach, the notion of protection is represented by an abstract entity, called a *protection domain*. Sharing is defined by *permission* relationships among protection domains. Applications can bind threads and data elements to different protection domains in order to dynamically implement different sharing relationships. The details regarding the behavior of the thread package can be found in [14].

2.3 Resource Allocation

We developed and implemented a framework for defining, analyzing, and controlling allocation of resources to mobile programs. The details of the approach can be found in [8, 7, and 6].

As part of this research, we developed a scheduling scheme and a set of algorithms for scheduling mobile programs. We describe them below:

- *Resource usage constraints:* We have defined a resource usage constraint specification language and a Java API that can be used by both site and mobile programs to specify how host-specific resources should be allocated. The language is similar to the access control language, except that it has been extended to allow a site to specify upper bounds, lower bounds, life time, and real-time usage constraints. The scheme, on the basis of mobile program requirements and host specifications, constructs a model of resource usage.

- *Construction of scheduling graph:* The scheme then partitions mobile programs into real-time and non-real-time programs. Mobile programs are grouped on the basis of the sites they are coming from, or on the basis of the kinds of resources the programs access.

- *Application of algorithms:* The scheme then applies a set of algorithms on the scheduling graph to schedule mobile programs. We have developed three different sets of algorithms: (i) an algorithm to enforce upper bounds and lifetime constraints; (ii) an algorithm to enforce share and priority constraints; and (iii) an algorithm to enforce real-time deadline based constraints. The scheduling scheme uses an algorithm composition policy for applying the algorithms on the scheduling graph.

- *Algorithm composition policy:* The schemes uses an algorithm composition policy to determine how conflicts among different resource usage constraints can be resolved. It implements a policy that always resolve conflicts in favor of security constraints.

We implemented the scheduling scheme within a simulation environment and the JVM (JDK version 1.1). We have also performed extensive performance analysis. The experimental results demonstrate that the scheme helps hosts to both protect and allocate CPU resources according to their preferences. The scheme effectively combines the scheduling algorithms in order to enforce both host and mobile program specified constraints.

2.4 Active Composition

A major weakness of our security model is the inability to modify the access control policies at runtime. The weakness arises primarily due to the manner in which we modify external programs and resources. The tool that we had developed enforces access control by editing an external program and resource code. However, any modifications in access control policies require that the already downloaded resource code and external program be modified during runtime. This is not currently possible in the JVM.

We started to look at the notion of active composition as a technique for implementing

dynamic access control and for optimizations that remove unnecessary security checks. Active composition involves allowing a program to modify its components at runtime. We can use this capability to re-implement security policies at runtime. We explored the following issues in this regard:

- Develop a model for supporting dynamic classes in the Java virtual machine.
- Design an implementation scheme for dynamic classes.
- Develop a dynamic access control mechanism and implement it using dynamic classes. We briefly describe a dynamic Java virtual machine and a dynamically reconfigurable security infrastructure below.

2.4.1 Dynamic Java Virtual Machine

We have developed a formal framework for composition, inheritance, and dynamic change in the Java environment. In our approach, Java programs can evolve by changing their components, namely classes, during their execution. Changes in a class also lead to changes in the instances of the class, thereby allowing evolution of both code and state. Our design is motivated by several design concerns: (i) Maintain type-safety of the Java programs since many of the Java's security mechanisms (for instance, separation of user and system name spaces, and protection of private data) depend on the assertion that an executing Java program is type safe. (ii) Maintain security properties of Java programs. Thus, we implement a security model that ensures that Java programs can dynamically modify only those resources to which they are authorized. We enforce this policy through name space separation and resource access control. Our work here involved the following:

- Development of a formal framework for composition, inheritance, and dynamic change in the Java environment.
- Design of an extensible class loader that allows programs to redefine its classes.
- Development of a security framework that insures that Java applets cannot modify protection mechanisms through the dynamic classes mechanism.
- Implementation of the dynamic classes mechanism.

The details regarding this work can be found in [9, 10]. While much of our work here was done to support a dynamic access control model, we believe that dynamic JVM can be used to several additional purposes, including runtime optimization, runtime code distribution and update, and hot updates. We are exploring several applications of dynamic classes.

2.4.2 Dynamic Access Control

We have developed a dynamic access control model in which security properties of a host can be defined and modified during runtime. Dynamic reconfiguration of security policies is needed in several instances, especially in complex and large distributed systems. While it is possible to represent many of the above conditions using the static security policy mechanisms, such representations may require anticipating all possible changes and specifying them. Such solutions may be awkward to represent, and may incur unnecessary overhead in many cases. We redesigned and re-implemented our access control mechanism to support a dynamic security infrastructure. The infrastructure includes the following:

- A declarative policy specification language for specifying access constraints.
- A meta-policy model that allows meta-representations of security policies. Within the meta-policy model, each security policy is represented as a first class object.
- A runtime meta-interface that can be used to inspect, add, delete, and modify security policies at runtime.
- An implementation where security policies are enforced through program transformation in which transposition code for protected resources are generated on-the-fly and integrated within the resource code before the modified resource is loaded in the Java virtual machine.

The details of the new policy model can be found in [4, 3].

2.5 Distributed Programming Environment

In addition to developing security techniques, we are also developing a distributed programming environment that will form the basis for integrating the security techniques. Unfortunately, the funding for the overall project ran out before we could fully develop the distributed programming environment and integrate all of the security techniques. We have continued to work on the programming environment, which is currently supported by a grant from NSF ITR. Below we briefly describe our approach.

The programming model supports an environment in which components of a program can freely migrate among distributed hosts. In this environment, both users and the underlying runtime system can initiate migration of components to adapt dynamically to changes in load, resource availability, resource distribution, component distribution, and other application-specific factors. Thus, the programming model supports mobility of code, data, and programs, and also for the dynamic control of both applications and servers. We implemented the model on top of Java JDK version 1.2. The details of this work can be found in [2].

3. Software Systems Produced

Over the course of the project, we have developed the following software systems:

- Access control specification compiler
- Java byte code binary editing tool
- Access constraint enforcement tool
- Safe threads package
- Resource scheduling simulator
- Modified JVM with hierarchical scheduling scheme
- Dynamic Java virtual machine
- Dynamic access control policy tool
- Distributed programming environment

4. Technology Transfer

University of California, Davis is filing a patent based on the work that was carried out during the project. The patent covers our research on dynamic systems. The university, through its technology transfer division, is in the process of creating a commercial relationship with companies such as Sun, HP, Oracle and IBM. This will lead to transfer of our ideas within the next generation of Java virtual machines.

5. Publications

[1] R. Pandey B. Hashii, S. Malarbarba and M. Bishop. Supporting reconfigurable security policies for mobile programs. Technical Report TR-2000-8, Department of Computer Science, University of California, Davis, 2000

[2] E. Barr, R. Pandey, and M. Haungs. Mage: A distributed programming model. In *Proceedings of the International Conference on Distributed Computing*, pages 303–312. IEEE, April 2001

[3] B. Hashii, S. Malabarba, R. Pandey, and M. Bishop. Dynamic security policies for mobile java programs. *Computer Networks*, (33):77–93, 2000

[4] B. Hashii, S. Malabarba, R. Pandey, and M. Bishop. Extensible security policies for mobile java programs. In *The Proceedings of the 9th International World Wide Web Conference*, pages 77–94, Amsterdam, May 2000. Elsevier

[5] B. Hashii, R. Pandey, S. Samorodin, and M. Lal. Securing Systems Against External Programs. *IEEE Internet Magazine*, 2(6):35–45, Nov/Dec 1998

[6] M. Lal and R. Pandey. CPU Resource Control for Mobile Programs. In *Proceedings of the First International Symposium on Agent Systems and Applications*, Oct 1999

[7] M. Lal and R. Pandey. A scheduling scheme for controlling allocation of CPU resources for mobile programs. Technical Report TR-1999-05, Department of Computer Science, University of California, Davis, 1999

[8] M. Lal and R. Pandey. A scheduling scheme for controlling allocation of CPU resources to mobile programs. *Journal of Autonomous and Multi-Agent Systems*, 2002

[9] S. Malabarba, R. Pandey, J. Gragg, E. Barr, and J. F. Barnes. Runtime support for type-safe dynamic java classes. In *The Proceedings of the European Conference on Object-Oriented Programming*, 2000

[10] S. Malarbarba, R. Pandey, J. Gragg, E. Barr, and J. Fritz Barnes. Support for type-safe dynamic java classes. Technical Report TR-2000-7, Department of Computer Science, University of California, Davis, 2000

[11] R. Pandey and B. Hashii. Providing fine-grained access control for java programs. In *Proceedings of the European Conference on Object-Oriented Programming*, 1999

[12] R. Pandey and B. Hashii. Providing fine-grained access control for java programs through binary editing. *Concurrency: Practice and Experience*, 2001

[13] R. Pandey, B. Hashii, and M. Lal. Secure execution of mobile programs. In *The Proceedings DARPA Information Survivability Conference and Exposition (DISCEX'00)*, 2000

[14] S. Samorodin and R. Pandey. Supporting flexible safety and sharing in multi-threaded environments. In *The Proceedings of the Run Time Systems Workshop, 2000*